2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

Recent Advances on GPU Computing in Operations Research

Vincent Boyer [1,2] , Didier El Baz [2]
[1] Graduate Program in Systems Engineering
Universidad Autónoma de Nuevo León, Mexico
[2] CNRS ; LAAS ; 7 avenue du colonel Roche, BP 54200 F-31031 Toulouse Cedex 4, France
Université de Toulouse, LAAS, F-31031 Toulouse France
Email: vincent.a.l.boyer@gmail.com, elbaz@laas.fr

***Abstract*—In the last decade, Graphics Processing Units (GPUs) have gained an increasing popularity as accelerators for High Performance Computing (HPC) applications. Recent GPUs are not only powerful graphics engines but also highly threaded parallel computing processors that can achieve sustainable speedup as compared with CPUs. In this context, researchers try to exploit the capability of this architecture to solve difficult problems in many domains in science and engineering. In this article, we present recent advances on GPU Computing in Operations Research. We focus in particular on Integer Programming and Linear Programming.**

***Keywords*-GPU Computing; Operations Research; Integer Programming; Linear Programming; Parallel Computing;**

I. Introduction

Originally designed for visualization purpose, graphics accelerators, that are many cores parallel architectures, have recently evolved towards powerful computing accelerators in collaboration with CPU for High Performance Computing (HPC) applications in science and engineering. In particular, they have been widely applied to signal processing and linear algebra.

We note that a device like the Tesla C2050 computing processor with Fermi architecture has 448 computing cores and 515 Gigaflops peak double precision floating point performance [1]. As a consequence, many computer manufacturers like Dell, HP, SGI and Bull are currently using Graphics Processing Units (GPUs) for acceleration purpose in the clusters and systems they propose. Moreover, some GPUs-based supercomputers like the Titan (17.6 Petaflops with NVIDIA K20 GPUs) in the USA and Tianhe-1A (2.57 Petaflop/s with C2050 GPUs) in China are in the Top 10 supercomputers ranking. One can quote also the Nebulea (1.27 Petalop/s with Intel X5650 processors and NVIDIA Tesla C2050 GPUs) in China, the Tusbame 2.0 in Japan and Roadrunner in the USA [2].

The exploitation of GPUs for HPC applications presents many advantages:

- GPUs are powerful accelerators since they have now hundreds of computing cores;
- GPUs are widely available and relatively chip;
- GPUs require less energy than other computing devices.

The recent interest in GPU computing and hybrid computing (which is a combination of CPU and GPU computing), is wide-spread. Almost all domains in science and engineering are concerned. We can quote for example astrophysics, seismic, oil industry and nuclear industry. Most of the time, GPUs lead to dramatic improvements in the solution time of practical problems.

It was quite natural for the Operations Research (OR) community whose field of interest is prolific in difficult problems to be attracted in GPU computing. In this paper, we present recent advances on GPU computing in this domain.

Section II deals with some aspects related to GPU programming. Application of GPUs to OR is presented in the sequel of the paper. In particular, contributions to Linear Programming (LP) are presented in Section III. Section IV deals with Integer Programming (IP). Conclusions and Challenges are presented in Section V.

II. GPU Computing And Hybrid Computing

Thanks to high-level shading languages like DirectX or OpenGL, graphics accelerators have started to be used for non-graphical applications in the early 2000s. By that time, problems like stock options pricing and protein folding have been solved on graphics accelerators showing noticeable speedup. The acronym GPU was then introduced by NVIDIA and people started to speak about General Purpose computing on the GPU (GPGPU). Programming graphics accelerators via graphics APIs turned out to be difficult since basic programming features were missing and programs were very complex (they had to be expressed in terms of textures, graphics concepts and shader programs). We note also that double precision floating point computation was not possible in the beginning.

GPUs were reimagined as highly threaded streaming processors when a new programming model extending C with data-parallel constructs was proposed by Ian Buck [3]. Concepts like kernels, streams and reduction operators were then introduced. A new compiler and runtime system permitted one to consider the GPU as a general-purpose processor in a high-level language leading also to substancial performance improvement.

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.45

GPU	# cores	Clock (GHz)	Memory (GB)
GeForce 7800 GTX	24	0.58	0.512
GeForce 8600 GTX	32	0.54	0.256
GeForce 9600 GT	64	0.65	0.512
GeForce GTX 260	192	1.4	0.9
GeForce GTX 280	240	1.296	1
GeForce GTX 285	240	1.476	1
GeForce GTX 295	240	1.24	1
GeForce GTX 480	480	1.4	1.536
Tesla C1060 [6]	240	1.3	4
T10 (Tesla S1070)	240	1.44	4
C2050 [1]	448	1.15	3

Table I
OVERVIEW OF NVIDIA GPUS QUOTED IN THE PAPER

The evolution of GPU's hardware that permits one to program more easily the device combined with the development in 2006 of Compute Unified Device Architecture, CUDA (a software and hardware architecture that enables the GPU to be programmed with some high level programming languages like C, C++ and Fortran) [4] or OpenCL (a framework for writing programs that are executed across heterogeneous platforms with CPUs, GPUs and other processors) [5] has fostered the popularity of GPU computing.

We recall that CUDA is a parallel computing platform and programming model designed and developed by NVIDIA. It permits one to increase computing performance by harnessing the power of the GPU. CUDA greatly simplifies GPU programming. One merely writes a serial codes intended to the CPU that calls parallel kernels defining the codes to be implemented by threads on the GPU cores. CUDA is based on a hierarchy of groups of threads and permits one to use synchronization barrier. CUDA functionalities are permanently extended in order to facilitate programming of GPUs. Among the many advantages of CUDA one can quote in particular: fast local memory that can be shared by a block of threads, double precision floating point arithmetics and more flexibility of coding than Graphics APIs. The recent CUDA 5.0 [4] was designed to facilitate the dynamic use of GPUs. Moreover, data transfers can now happen via high-speed network directly out of any GPU memory to any other GPU memory in any other cluster without involving assistance of the CPU.

Table I displays the characteristics of several GPUs considered in the sequel. Up to very recently, the Fermi architecture represented the last generation of NVIDIA GPU architectures [7]. With 3.0 billion transistors, the Fermi architecture features up to 512 CUDA cores; it is built around a scalable array of multithreaded Streaming Multiprocessors (SM). The 512 CUDA cores are organized in 16 SMs. Typically, a Fermi multiprocessor consists of 32 Scalar Processors cores. A CUDA core executes a floating point or integer instruction per clock for a thread. Each CUDA core or processor has a fully pipelined integer Arithmetic Logic Unit (ALU) and Floating Point Unit (FPU). The Fermi architecture implements the IEEE 754-2008 floating-point standard, providing the Fused Multiply-Add (FMA) instruction for both single and double precision arithmetic. Fermi architecture has a two-level distributed thread scheduler. The GigaThread global scheduler distributes thread blocks to SM thread schedulers. Concurrent threads are created, managed and run by the SM thread scheduler without overhead. This permits one in particular to implement data parallelism. NVIDIA has emphasized on the Single Instruction, Multiple Threads (SIMT) parallel programming model. The reduction in computation time resulting from exploiting GPUs for data parallel applications can be dramatic. However, we note that maximizing effective memory bandwidth of the GPU and having good thread occupancy, i.e., giving sufficient work to the SMs as well as ensuring coalesced memory accesses for all cores of a given SM (in particular by avoiding divergent branches) is particularly important in order to obtain noticeable speedup with a GPU. It is also important to note that thanks to CUDA or OpenCL, multi-GPU computing is possible, i.e., the possibility to exploit several accelerators in a unique application. This leads to massive parallelism and the use GPUs for HPC applications.

Finally, we note that some libraries like the NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS) [8] have been developped to help programmers to solve large scale problems on the GPU. The cuBLAS library is a GPU-accelerated version of the complete standard BLAS library.

III. LINEAR PROGRAMMING

In linear programming, the variables are nominally allowed to take a continuous range of values. The standard form of such a problem can be formulated as follows:

$$\min\ p^T x, \tag{1}$$

$$s.t.\ Ax = b, \tag{2}$$

$$x \geq 0, \tag{3}$$

where x is the vector of real numbers to be determined, A is a matrix whose entries are fixed real contants, b and p are two vectors of fixed real constants.

The simplex method has been designed by George Dantzig [9] and different variants of the method have been proposed in the literature for the solution of problem (1) - (3). A complete review of the simplex method and its variants can be found in [10]. Basically, the simplex algorithm starts from a feasible solution at a vertex of the polytope and tries to improve the solution while preserving feasibility until optimality is reached. In this section, we focus on several approaches that have been inplemented on the GPU. Table II summarizes the contributions in the literature.

A. The Simplex Tableau

In the simplex tableau algorithm, data are organized in a tableau; pivoting operations are then applied until optimality.

Algorithm	Year	Authors
The Simplex Tableau	2011	Lalami et al. [11], [12]
	2011	Meyer et al. [13]
The Revised Simplex	2005	Greef [14]
	2009	Spampinato et al. [15]
	2010	Bieling et al. [16]
The Interior Point Method	2008	Jung and O'Leary [17]

Table II
LITERATURE OVERVIEW IN LINEAR PROGRAMMING AND GPUS

This data structure is well suited to the GPU architecture: it naturally tends to ensure efficient coalesced memory access and does not need extra efforts for memory optimization.
In 2011, Lalami et al. have proposed a GPU implementation of the simplex tableau algorithm via CUDA 2.3 in [11]. This implementation has been extended to the multi-GPU context in [12]. The same year, Meyer et al. have proposed a different multi-GPU implementation of the simplex tableau algorithm in [13]. Both papers have dealt with a complete implementation of the simplex algorithm on the GPUs including the pivoting and the selection of the entering and leaving variables in order to avoid extra communication between the CPU and the GPUs. Multi-GPU computing relies on problem decomposition. Several decomposition schemes can be adopted. An horizontal decomposition distributes the constraints on the different GPUs. A vertical decomposition distributes the variables of the LP problem on the GPUs. Finally, one may consider also tiles. The choice of a decompositon scheme has important consequences on the resulting communication pattern and multi-GPU efficacy. A decomposition based on tiles may appear scalable; it nevertheless necessitates many communications between GPUs. In [13], the authors have adopted a vertical decomposition in order to have less communication between GPUs. An horizontal decomposition has been adopted in [12]. The simplex tableau has been decomposed into parts that are assigned to the different GPUs. An horizontal decomposition of the simplex tableau presents the advantage to facilitate the parallel processing of the entering variable column and the ratio column (the leaving variable line is computed by only one GPU at each iteration). The drawback of this decomposition is to duplicate data. In [12], each GPU updates only a part of the tableau. The work of each GPU is managed by a distinct CPU thread. More precisely, CPU threads are in charge of launching the kernels of the simplex algorithm on each GPU, synchronising the work among the different GPUs and sharing the results. This approach presents the advantage to maintain the context of each CPU thread all along the application, i.e., CPU threads are not killed at the end of each simplex iteration. As a consequence, communications tend to be minimized.

Lalami et al. [12] have used a server with Intel Xeon E5640 2.66GHz processor and two NVIDIA C2050 GPUs. They have considered instances with up to 27,000 variables and 27,000 constraints. The computational tests have shown significant speedups. For example, a reduction of the computation time by a factor of 24.5 with two GPUs has been observed for the largest instance. Meyer et al. [13] have used a system with two Intel Xeon X5570 2.93GHz processor and one NVIDIA Tesla S1070 computing system featuring T10 GPUs. They have solved instances with up to 25,000 variables and 5,000 constraints and outperformed the open-source solver CLP [18] of the COIN-OR project.

B. The Revised Simplex

In the revised simplex method, only data that are relative to the basic variables are stored in a tableau. This allows a lower memory requirement than in the simplex tableau algorithm and suggests a reduction in complexity. All operations of the simplex method can be seen as matrices operations.

Greeff [14] was the first to accelerate the revised simplex method via GPU. He has achieved a speedup of 11.5 as compared to an identical CPU implementation. Most of the GPU computing drawbacks encountered by Greef in 2005 have been addressed since.

Later, Spampinato et al. [15] have tried to take benefit from the advances in the linear algebra library cuBLAS [8]. They have used a system with Intel Core 2 Quad 2.83GHz processor and NVIDIA GeForce GTX 280 GPU and have reported a reduction in solution time by factor of 2.5 for problems with 2,000 variables and 2,000 constraints when compared to the ATLAS-based solver [19].

More recently, Bieling et al. [16] have proposed an implementation of the revised simplex method which includes some algorithmic optimization, i.e., the steepest-edge heuristic to select the entering variables [20] and an arbitrary bound process in order to select the leaving variables. The authors have reported a reduction in computation time by a factor of 18 for instances with 8,000 variables and 2,700 constraints on a system with Intel Core 2 Duo E8400 3.0 GHz processor and NVIDIA GeForce 9600 GT GPU when compared to results obtained with the GLPK solver [21].

C. The Interior Point Method

Interior point methods for solving problem (1) - (3) have been considered for implementation on systems with GPUs. We recall that these methods, also referred to as barrier methods, reach the optimal solution of the LP by traversing the interior of the feasible region.

Jung and O'Leary [17] have proposed a mixed precision hybrid algorithm for solving LP using a primal-dual interior point method. The algorithm is based on the rectangular-packed matrix storage scheme and uses the GPU for computationally intensive tasks like matrix assembly, Cholesky factorization and forward and back substitution.

The hybrid algoritm was carried out on a system with Intel Xeon 3.0GHz processor and NVIDIA GeForce 7800 GTX GPU. Instances with up to 4,000 variables and 1,000

constraints have been considered; nevertheless it turns out that the proposed hybrid algorithm has not clearly outperformed the sequential version on CPU due to data transfer cost and communication latency.

IV. Integer Programming

Integer Programming problems, IP, occur for example in transportation, planning and logistics. In the standard form, IP problems can be expressed as follows:

$$max\ p^T x, \tag{4}$$

$$s.t.\ Ax = b, \tag{5}$$

$$x \geq 0, \tag{6}$$

$$x\ integer. \tag{7}$$

where the entries of A, b and p are integer constants. Many IP problems are NP-hard. Solution via dynamic programming and Branch and Bound is often considered in the literature. The resulting data structure is often irregular; thus, it is not well suited to GPU computing and solving IP with the help of GPUs is in many case a challenge. We can find two types of parallel approaches:

- either IP is entirely solved on GPU(s) through a specific or adapted parallel algorithm;
- or GPUs are used to accelerate only the most time consuming activities or parts of codes (hybrid algorithms).

To the best of our knowledge, four types of problems have been studied in the GPU literature: Knapsack Problems (KP), Scheduling Problems (SP), Assignment Problems (AP) and Travelling Salesman Problems (TSP). The proposed approaches and computational results are presented in the sequel. The literature is summarized in Table III.

A. Knapsack Problems

1) Dynamic programming: The solution of KP via a hybrid dense dynamic programming algorithm implemented with CUDA 2.0 has been considered in [22]. At each step, computations in the loop that processes the classical Bellman's dynamic programming recursion (which is time consuming) have been implemented in parallel on the device.

A data compression technique has also been proposed in order to deal with the high memory requirement of the dynamic programming method. This technique has permitted the authors to reduce the memory occupancy needed to reconstruct the optimal solution and the amount of data transferred between the host and device.

Computational experiments have been carried out on a system with Intel Xeon 3.0 GHz and NVIDIA GTX 260 GPU. Randomly generated correlated problems with up to 100,000 variables have been considered. We note that dense dynamic programmic is known to be suited to correlated instances. Computational results have shown that these problems can be solved within relatively small computing time via GPU (only few hundred seconds) and memory occupancy. Moreover, a reduction in computation time by a factor of 26 has been observed for instances with more than 40,000 variables. We note that the reduction in matrix size is better when the size of the problem is increased, resulting in a more efficient compression while the overhead does not exceed 3% of the overall processing time.

The contribution in [22] has been further extended in [23], where a multi-GPU hybrid implementation via CUDA 2.3 of the dense dynamic programming method has been proposed. The approach is well suited to the case where a CPU is connected to several GPUs. The solution presented in [23] is based on multithreading and the concurrent implementation of kernels on GPUs; each kernel being associated with a given GPU and managed by a CPU thread; the context of each host thread being maintained all allong the application, i.e., host threads are not killed at the end of each dynamic programming step. This technique tends also to reduce data exchanges between host and device. A load balancing procedure has been implemented in order to maintain efficiency of the parallel algorithm.

Computational experiments have been carried out on a machine with Intel Xeon 3 GHz processor, 1 GB memory and Tesla S1070 computing system. Strongly correlated problems with up to 100,000 variables have been considered. Preliminary results have shown a reduction in computation time by a factor of 14 with one GPU and 28 with two GPUs (without data compression techniques).

2) Branch and Bound: A hybrid Branch and Bound algorithm has been proposed in [25] for the solution of the knapsack problem. The GPU is used only when the number of Branch and Bound nodes is important. Computation is performed on the CPU when the number of nodes is under a certain threshold. Elimination of nonpromising nodes and concatenation of the list of nodes is always performed on the CPU.

Dimension and capacity of the problem are stored in the constant memory of the GPU since they do not change during the solution of the problem. Weights and profits of items are stored in the texture memory of the GPU that is larger than the constant memory; both memories presenting low latency. When in use, the GPU takes care of the separation phase, i.e., creation of new nodes in parallel, it also computes bounds in parallel and the best lower bound *via atomicMax* operation. Finally, the GPU performs in parallel bounds comparison and nonpromising nodes labelling (the later task concerns nodes with an upper bound smaller than the best lower bound).

Computational tests have been carried out on a system with Intel Xeon E5640 2.66GHz processor and NVIDIA C2050 GPU. Reduction in computation time by a factor of 20 has been observed in [25] for strongly correlated problems with 500 variables. The reduction in computation time has been further improved by a factor of 52 in [26],

Problem	Algorithm	Year	Authors
KP	Dynamic Programming	2011-2012	Boyer et al. [22], [23]
	Branch and Bound	2012	Boukedjar et al. [24]
		2012	Lalami et El Baz [25]
		2012	Lalami [26]
	Genetic Algorithm	2012	Pedemonte et al. [27]
SP	Tabu Search	2008	Janiak et al. [28]
		2011	Czapiński et al. [29]
		2011	Luong et al. [30]
		2012	Bukata [31]
		2013	Bukata and Šucha [32]
	Branch-and-Bound	2012	Chakroum et al. [33], [34]
		2012	Melab et al. [35]
	Genetic Algorithm	2011	Zajíček and Šucha [36]
		2011	Nesmachnow et al. [37]
		2013	Pinel et al. [38]
AP	Tabu Search	2010	Luong et al. [39]
		2011	Luong [40]
	Genetic Algorithm	2009	Tsutsui et Fujimoto [41]
		2010	Soca et al. [42]
		2011	Tsutsui et Fujimoto [43]
	Deep Greedy Switching	2011	Roverso et al. [44]
TSP	Ant Colonies	2007	Catala et al. [45]
		2009	Li et al. [46]
		2009	You [47]
		2011	Cecilia et al. [48]
	Max-Min Ant System	2009	Jiening et al. [49]
		2009	Bai et al. [50]
		2010	Fu et al. [51]
	Genetic Algorithm	2011	Chen et al. [52]
	Immune Algorithm	2009	Li et al. [53]
	Tabu Search	2008	Janiak et al. [28]

Table III
LITERATURE OVERVIEW IN INTEGER PROGRAMMING AND GPU COMPUTING

for strongly correlated problems with 1,000 variables and some optimization in access to the GPU memory. The computational results have shown that the more difficult a problem is, the larger the number of Branch and Bound nodes and the more remarkable the reduction in time due to GPU accelerator. The reader is also referred to [24] for another contribution to this field.

3) Genetic Algorithm: Pedemente, Alba and Luna have proposed in [27] a genetic algorithm specially designed to run on GPU. The algorithm called Systolic Genetic Search (SGS) is based on the model of systolic computation, i.e., the synchronous circulation of solutions through a grid of processing units. At each iteration, the crossover and mutation operators, the fitness function evaluation and the elitist replacement are carried out on the GPU. The exchange of directions operator can additionally be applied on the GPU. The five kernels quoted above are invoked by the host.

Experiments have been carried out on a system with Pentium D 3.0 GHz processor, 2 GB RAM and NVIDIA GeForce GTX 480 GPU. Problems without correlation and up to 1,000 variables have been considered. Experimental results have shown that the SGS method produces solutions of very good quality. GPU and CPU versions of SGS have been carried out with the same seeds so that experimental results were exactly the same. Numerical results have shown that the reduction in computation time ranges from 5.09 to 35.7 times according to the size of the considered instances.

B. Scheduling Problems

1) Branch and Bound: The solution of the Flow-shop Scheduling Problem (FSP) via parallel Branch and Bound methods using GPU has been studied by several authors.

In [33], a selection operator based on the best-first strategy is used until the work pool reaches a given size. Then, a selection operator based on the depth-first strategy is used. This technique permits one to provide enough work to the GPU. The evaluation of bounds that is time consuming was performed in parallel in [33]; an original technique was also proposed to avoid divergent threads in a warp resulting from conditional branches.

Computational experiments have been carried out via CUDA 4.0 on a system with Intel Xeon E5520 2.27 GHz bi-processor and NVIDIA C2050 computing system. Some instances of flow-shop problems proposed by Taillard (see [54]) that range from twenty jobs on twenty machines to two hundred jobs on twenty machines have been considered. Maximum speedup factor of 77 has been observed for instances with two hundred jobs on twenty machines as compared with a sequential version.

The parallel application of branching, bounding, selection and elimination operators has been considered in [34] as well

as exploiting higher parallelism via workload distribution on a multi-GPU testbed. Workloads have been equally splitted into as many groups as there are GPUs in the system and an equal number of CPU threads has been created.

In [34], computational experiments have been carried out via CUDA 4.0 on a system with Intel Xeon E5520 2.27 GHz bi-processor and NVIDIA Tesla S1070 computing system. The considered problems range from twenty jobs on ten machines to two hundred jobs on twenty machines. Instances have been solved on one GPU 11 to 78 times faster than with a single CPU core. Morover, a maximum speedup factor of 105 has been observed with two GPUs.

Reference is also made to [35] for a study on a parallel Branch and Bound method using GPU whereby the evaluation of lower bounds is made on the device while generation of subproblems, i.e., elimination, selection and branching operations is implemented on the host.

2) Genetic algorithms: Zajíček and Šucha have studied a parallel island-based Genetic Algorithm (GA) for the solution of the FSP in [36]. According to the proposed homogeneous model, all computations were carried out on the GPU in order to reduce communication between CPU and GPU. The authors have implemented a GA whereby islands are essentially used for migration of individuals, i.e., specific solutions among subsets of solutions, the so-called populations. Evaluations, mutations and crossovers of solutions in the same subset of solutions were performed in parallel and independently of other populations.

Experiments were performed on a system with AMD Phenom II X4 945 3.0 GHz processor and NVIDIA Tesla C1060 GPU. Some instances with one hundred activities and five machines were solved 110 faster than with the AMD CPU.

Nesmachnow and Canabé have considered the solution of the Heterogenous Computing Scheduling Problem. A parallel implementation of the Min-Min heuristic on GPU, whereby the evaluation of the criteria for all machines is made in parallel on the GPU for each unassigned task, was proposed in [37] (a parallel version of the Sufferage heuristic was also studied in the paper).

Computational experiments were carried out on a system with Dell Xeon E5530 2.4 GHz processor and NVIDIA C1060 GPU. Reduction of computational time by a factor of 5 has been obtained for parallel Min-Min (5.5 for parallel Sufferage).

Pinel et al. have considered the parallel solution of scheduling of independant tasks problems in [38]. They have also proposed implementations on GPU of the Min-Min heuristic and GraphCell a parallel cellular genetic algorithm (two new parallel recombination operators are also proposed in the paper).

Computational experiments have been carried out on a system with Intel Xeon E5440 2.83 GHz processor and NVIDIA Tesla C2050 GPU. We note that significant speedup has been obtained for the GPU version of the Min-Min heuristic.

3) Tabu Search: Czapiński and Barnes have implemented a Tabu Search (TS) metaheuristic method for the solution of FSP via GPU in [29].

The TS method was carried out on a system with Intel Xeon 3.0 GHz processor with 2GB memory and NVIDIA Tesla C1060 GPU. The authors have claimed that their implementation is 89 times faster than the CPU version.

In [30], Luong, Melab and Talbi have studied the implementation on GPU of an aggregated TS method for the FSP. Their paper deals more generaly with the implementation on GPU of multiobjective local search algorithms. The generation of neighborhood was done on GPU in order to reduce data transfers; several representations have been considered (see also [40] and [55]).

Experiments have been carried out on two systems:

- a system with Xeon 3.0 GHz processor and GTX 285 GPU;
- a system with Core i7 3.2 GHz processor and GTX 480 GPU.

The considered problems range from twenty jobs and ten machines to two hundred jobs and twenty machines. The observed maximum speedup was 10 times for the first system and 16 times for the second system.

We note that Pareto local search algorithms have also been studied in [30]. The same instances have been considered and the observed maximum speedup was 9.4 times for the first system and 15.7 times for the second system.

Bukata and Šucha have subsequently presented a parallel TS method for the Resource Constrained Project Scheduling Problem (RCPSP) according to the proposed homogeneous model whereby all computations are performed on the GPU (see [32]). The Simple Tabu List implementation (with constant algorithmic complexity) was specially designed for the GPU, a new parallel algorithm for schedule evaluation was proposed and parallel reductions were applied.

Experiments have been carried out via CUDA 3.2 on a server with Intel Xeon E5640 2.66 GHz processor, 12 GB memory and NVIDIA Tesla C2050 GPU. Solution interchange was performed via the global memory where the global best solution and working set were stored. Simple Tabu lists were also stored in the global memory. Local memory was used for resource arrays and activities start time values. J120 instances with 600 projects and 120 activities have been used for performance comparison. Experimental results have shown that the GPU is able to perform the same number of iterations 55 times faster than the CPU in average (see also [31]). Experiments have also shown that the parallel TS method outperforms the CPU version on what concerns the quality of solutions that is comparable to the one obtained with efficient metaheuristics in the literature (see [32]).

C. Assignment Problems

1) Deep Greedy Switching: Roverso et al. have proposed a GPU implementation of the Deep Greedy Switching (DGS) heuristic for the solution of the Linear Sum Assignment Problem (LSAP) in [44]. Classically, agents have to be assigned to an equal number of jobs while maximizing the total benefit. Basically, the DGS heuristic starts from a random initial solution and moves to better solutions by considering a neighborhood resulting from a restricted 2-exchange. Each agent tries to find the best solution from a given neighborhood. The improvement, or difference, in the objective function between the current and new solution, called agent difference evaluations and job difference evaluations, that are computationally expensives have been carried out in parallel on the GPU.

Computational experiments have been carried out on a system with Core 2 Duo 2.4 GHz processor, 4 GB memory and NVIDIA GTX 295 GPU. Randomly generated instances with up to 9744 jobs have been considered. Reduction of computation time by a factor of up to 27 has been observed.

2) Tabu Search: We note also that Luong et al. have proposed a hybrid Tabu Search method for the 3-dimensional Quadratic Assignment Problem (QAP) in [39]. The basic TS algorithm runs on the CPU. Evaluation and neighborhood generation that are time consumming run on the GPU.

Computational tests were performed on a system with Intel Core Duo 3.0 GHz processor and NVIDIA 8600 GT GPU. A reduction of computation time by a factor of 4 was obtained.

3) Genetic Algorithm: Tsutsui and Fujimoto have proposed to solve the QAP via a parallel Ant Colony Optimization method (ACO) implemented on GPU in [43]. They have considered 2-opt local search whereby moves can be divided into two groups that can be computed in parallel by blocks of threads. Pheromone update and sampling are also carried out on the device.

Computational tests have been carried out on a system with Intel Core i7 965 3.2 GHz processor and NVIDIA GeForce GTX 480 GPU. Real life like instances and randomly generated instances of the QAPLIB library whose size ranges from 50 to 150 have been considered. Speedups of 24.6 times have been observed as compared with a sequential version of the method implemented on the CPU (reference is also made to [41]).

Soca et al. [42] have proposed a framework for automatic implementation of parallel cellular genetic algorithms on GPU.

D. Travelling Salesman Problems

Giving a set of cities, the TSP [56] involves finding the shortest route that visits each city exactly once. It is a well-known NP-hard optimization problem and is used as a standard benchmark for many heuristic algorithms. The instances of the TSPLIB library [57] are generally used as benchmark instances for computational tests.

In this section, we focus on the ant colony approaches which have been given a particular attention in the literature for the solution of the TSP via GPUs. Ant colonies [58] are population-based metaheuristics for solving optimization problems. They use artificial ants to construct solutions by considering pheromone trails that reflect the search procedure.

The implementation on GPU of some other metaheuristics has been investigated for the TSP. The reader is referred to the work of Janiak et al. on Tabu Search [28], Li et al. on Immune Algorithm [53] and Chen et al. on Genetic Algorithm [52]. To the best of our knowledge, no exact approach has been addressed in the literature.

1) Ant Colonies Optimization (ACO): The first proposed work in this area was due to Catala et al. in 2007 (see [45]); the solution of the orienteering problem, also known as the selective travelling salesperson problem (see [59]), was considered. The results have shown that the proposed approach that was implemented on a single GeForce 6600 GT GPU stood competitive with a parallel ACO running on a GRID with up to 32 nodes.

In 2009, Li et al. [46] and You et al. [47] have proposed GPU implementations of the ACO method with adequate memory management.

The implementation of Li et al. is based on a fine-grain model whereby ants are assigned to one thread.

You et al. have focused on the construction phase of the ACO whereby each GPU thread builds a route for one ant. Computational experiments have been carried out on a system with Intel Core 2 Duo 2.20GHz processor and GeForce 8600GT GPU. Problems with up to 800 cities have been considered and a maximum speedup of 20 has been observed.

In 2011, Cecilia et al. [48] have presented new results on the parallelization of the ACO method on GPU. Different strategies for the implementation of both stages of the ACO algorithm on the GPU, i.e., the construction phase and the pheromone update are discussed in the paper. The authors have proposed to assign one ant to each block of threads Moreover, each thread represents a city or a set of cities the ant may visit. This strategy is followed in order to overcome some drawbacks encountered by former methods in the literature. For the pheromone update, the authors have proposed a scatter-to-gather transformation (see [60]) which avoids the use of atomic operations on GPU.

Computational experiments have been carried out on a C1060 GPU. Speedup factors of 25 have been reported for some instances with 2,396 cities.

2) Max-Min Ant System: In 2009, Jiening et al. have proposed an implementation of a variant of the ACO method: the Max-Min Ant System (MMAS) [49]. In this implementation, the tour construction stage is carried out on GPU,

and the shortest path is computed on the CPU.

Computational experiments have been carried out on a system with AMD 2.79Hz processor and NVIDIA Quadro Fx 4500 GPU. On a typical test instance with thirty cities, a reduction of computation time by a factor of 1.4 was observed.

In 2009, Bai et al. have discussed the implementation of all phases of MMAS on GPU [50]. For the tour construction stage each ant colony is assigned to a thread block, whilst for the pheromone update, each city corresponds to one thread.

Computational experiments have been carried out on a system with AMD Athlon Dual Core Processor 3600+ and NVIDIA GeForce 8800 GTX GPU. Processing time twice as fast as with CPU have been reported for instances with 400 cities.

More recently, Fu et al. have proposed in [51] an implementation of MMAS on GPU that makes use of the Jacket toolbox which connects MATLAB to GPU (see [61]). Ants share only one pseudorandom number matrix, one pheromone matrix, one taboo matrix and one probability matrix in order to reduce communication between CPU and GPU. Furthermore, a variation of the traditional roulette wheel selection (the All-In-Roulette which is well suited to the GPU architecture) has been used.

Computational tests have been carried out on a system with Intel i7 3.3GHz processor and Tesla C1060 GPU. Speedups of 400 on GPU as compared with the sequential algorithm have been observed for an instances with up to 1,002 cities.

E. Frameworks for the design of combinatorial algorithms on GPU

We conclude this section with some optimization frameworks that have been designed for or extended to the transparent deployment of metaheuristics on GPU.

We can quote for example the ParadisEO-GPU framework developed by the Dolphin team at University of Lille 1 (see [40]). ParadisEO is a portable template-based C++ library dedicated to the design of metaheuristics for optimization problems. ParadisEO-GPU extends a module which is a coupling between ParadisEO and CUDA.

Experiments carried out for the Quadratic Assignment Problem with the ParadisEO-GPU framework and the Tabu Search method have shown small performance degradation as compared with the version described in subsection IV-C2.

We have seen in subsection IV-C3 that the Pugace framework [42] was also designed for the automatic implementation of cellular evolutionary algorithms on GPU.

V. Conclusions and Challenges

In this paper, we have concentrated on a hot topic: GPU computing in the field of Operational Research. We have presented important contributions to Integer Programming and Linear Programming. We note that many OR problems have been considered in the literature and that significant speedups have been obtained for several exact methods or heuristic algorithms thanks to the GPUs. At this point, it is important to note that one cannot establish a quantitative comparison between the proposed approaches since they have led to implementations on several generations of GPUs.

For many applications including OR problems, the future of GPU computing seems very promising. The new NVIDIA Kepler architecture features teraflops of integer, single precision, and double precision performance and high memory bandwidth [62]. Moreover, programming tools like CUDA and OpenCL (or the recent OpenACC [63]) always tend to facilitate programming and improve efficiency of this type of architecture by hiding programming difficulties. We note in particular that OpenACC is a set of high-level pragmas that enables C/C++ and Fortran programmers to exploit highly parallel processors with much of the convenience of OpenMP. In this case, pragma are code annotations which inform the compiler of structured loop or succeeding block of code as a good candidate for parallelization.

We believe that OR industrial codes in the future will be able take a great benefit from GPUs and to propose very attractive and fast solutions to their users. An important challenge remains in the exact solution of industrial problems of significant size via GPUs.

Acknowledgments

Dr. Didier El Baz thanks NVIDIA for support through Academic Partnership.

References

[1] NVIDIA, *TESLA C2050 and TESLA C2070 Computing processor board*. NVIDIA Corporation, 2010.

[2] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. (2012, Nov.) The top500 list. [Online]. Available: http://www.top500.org/

[3] I. Buck, *Stream Computing on Graphics Hardware*. Ph.D. Thesis, Stanford University, 2006.

[4] NVIDIA. (2013) Cuda 5.0. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[5] Khronos. (2013) Opencl. [Online]. Available: http://www.khronos.org/opencl/

[6] NVIDIA, *GPU Computing Solutions NVIDIA TESLA*. NVIDIA, 2009.

[7] ——, *Whitepaper NVIDIA's Next Generation CUDA Compute Architecture: Fermi*. NVIDIA Corporation, 2009.

[8] ——. (2013) cuBLAS. [Online]. Available: https://developer.nvidia.com/cublas

[9] G. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," in *Activity Analysis of Production and Allocation*. Wiley and Chapman-Hall, 1951, pp. 339–347.

[10] A. Schrijver, *Theory of integer and linear programming*. Wiley, Chichester, 1986.

[11] M. Lalami, V. Boyer, and D. El Baz, "Efficient implementation of the simplex method on a CPU-GPU system," in *25th IEEE International Parallel and Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW), Workshop PCO'11*, may 2011, pp. 1999 –2006.

[12] M. Lalami, D. El Baz, and V. Boyer, "Multi GPU implementation of the simplex algorithm," in *13th IEEE International Conference on High Performance Computing and Communications (HPCC 2011)*, sept. 2011, pp. 179 –186.

[13] X. Meyer, P. Albuquerque, and B. Chopard, "A multi-GPU implementation and performance model for the standard simplex method," in *Euro-Par 2011*, 2011.

[14] G. Greeff, "The revised simplex algorithm on a GPU," *Univ. of Stellenbosch, Tech. Rep*, 2005.

[15] D. Spampinato and A. Elster, "Linear optimization on modern GPUs," in *2009 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2009*, may 2009, pp. 1 –8.

[16] J. Bieling, P. Peschlow, and P. Martini, "An efficient GPU implementation of the revised simplex method," in *24th IEEE International Parallel Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW 2010)*, april 2010, pp. 1 –8.

[17] J. Jung and D. O'Leary, "Implementing an interior point method for linear programs on a CPU-GPU system," *Electronic Transactions on Numerical Analysis*, vol. 28, pp. 174–189, 2008.

[18] COIN-OR. (2013) CLP. [Online]. Available: https://projects.coin-or.org/Clp

[19] (2013) ATLAS. [Online]. Available: math-atlas.sourceforge.net/

[20] D. Goldfarb and J. Reid, "A practicable steepest-edge simplex algorithm," *Mathematical Programming*, vol. 12, no. 1, pp. 361–371, 1977.

[21] GNU. (2013) GLPK. [Online]. Available: http://www.gnu.org/software/glpk/

[22] V. Boyer, D. El Baz, and M. Elkihel, "Dense dynamic programming on multi GPU," in *19th International Conference on Parallel, Distributed and network-based Processing (PDP 2011)*, feb. 2011, pp. 545 –551.

[23] ——, "Solving knapsack problems on GPU," *Computers and Operations Research*, vol. 39, no. 1, pp. 42 – 47, 2012.

[24] A. Boukedjar, M. Lalami, and D. El Baz, "Parallel branch and bound on a CPU-GPU system," in *20th International Conference on Parallel, Distributed and network-based Processing (PDP 2012)*, feb. 2012, pp. 392 –398.

[25] M. Lalami and D. El Baz, "GPU implementation of the branch and bound method for knapsack problems," in *26th IEEE International Parallel and Distributed Processing Symposium, Workshops and PhD Forum (IPDPSW), Workshop PCO'12*, may 2012, pp. 1769 –1777.

[26] M. Lalami, *Contribution à la résolution de problèmes d'optimisation combinatoire : méthodes séquentielles et parallèles*. Ph. D. Thesis, Université Paul Sabatier, Toulouse, 2012.

[27] M. Pedemonte, E. Alba, and F. Luna, "Towards the design of systolic genetic search," in *26th IEEE International Parallel and Distributed Processing Symposium, Workshops and PhD Forum (IPDPSW 2012), Workshop PCO'12*, may 2012, pp. 1778 –1786.

[28] A. Janiak, W. Janiak, and M. Lichtenstein, "Tabu Search on GPU," *Journal of Universal Computer Science*, vol. 14, no. 14, pp. 2416–2427, jul 2008.

[29] M. Czapiński and S. Barnes, "Tabu Search with two approaches to parallel flow shop evaluation on CUDA platform," *Journal of Parallel and Distributed Computing*, vol. 71, pp. 802 – 811, 2011.

[30] T. V. Luong, N. Melab, and E. Talbi, "GPU-based approaches for multiobjective local search algorithms. a case study: the flowshop scheduling problem," *Evolutionary Computation in Combinatorial Optimization*, pp. 155–166, 2011.

[31] L. Bukata, *GPU algorithms design for resource constrained project scheduling problem*. Ph .D. Thesis, Czech Technical University in Prague, 2012.

[32] L. Bukata and P. Šucha, "A GPU algorithm design for resource constrained scheduling problem," in *Proceedings of the 21st Conference on Parallel, Distributed and networked-based Processing (to appear), Belfast*, 2013.

[33] I. Chakroun, M. Mezmaz, N. Melab, and A. Bendjoudi, "Reducing thread divergence in a GPU-accelerated branch-and-bound algorithm," *Concurrency and Computation: Practice and Experience*, 2012.

[34] I. Chakroun and N. Melab, "An adaptative multi-GPU based branch-and-bound. a case study: the flow-shop scheduling problem," *CoRR*, 2012.

[35] N. Melab, I. Chakroun, M. Mezmaz, and D. Tuyttens, "A GPU-accelerated branch-and-bound algorithm for the flow-shop scheduling problem," in *2012 IEEE International Conference on Cluster Computing (CLUSTER)*, sept. 2012, pp. 10–17.

[36] T. Zajíček and P. Šucha, "Accelerating a flow shop scheduling algorithmon the GPU," in *Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP)*, 2011.

[37] S. Nesmachnow and M. Canabé, "GPU implementations of scheduling heuristics for heterogeneous computing environments," in *XVII Congreso Argentino de Ciencias de la Computación*, 2011.

[38] F. Pinel, B. Dorronsoro, and P. Bouvry, "Solving very large instances of the scheduling of independent tasks problem on the GPU," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 101 – 110, 2013.

[39] T. V. Luong, L. Loukil, N. Melab, and T. E. L., "A GPU-based iterated tabu search for solving the quadratic 3-dimensional assignment problem," in *Computers Systems and Applications (AICCSA)*, 2010.

[40] T. Luong, *Métaheuristiques parallèles sur GPU*. Ph. D. Thesis, Université Lille 1, 2011.

[41] S. Tsutsui and N. Fujimoto, "Solving quadratic assignment problems by genetic algorithms with GPU," in *Proceedings of the 11th Annual Conference compaanion on Genetic and Evolutionary computation Conference, GECCO'09, New York City, USA*, 2009, pp. 2523–2530.

[42] N. Soca, J. Blengio, M. Pedemonte, and P. Ezzatti, "Pugace, a cellular evolutionary algorithm framework on GPUs," in *2010 IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 1–8.

[43] S. Tsutsui and N. Fujimoto, "Fast QAP solving by ACO with 2-opt local search on a GPU," in *2011 IEEE Congress on Evolutionary Computation (CEC)*, june 2011, pp. 812 –819.

[44] R. Roverso, A. Naiem, M. El-Beltagy, S. El-Ansary, and S. Haridi, "A GPU-enabled solver for time-constrained linear sum assignment problems," in *7th International Conference on Informatics and Systems (INFOS)*, 2011, pp. 1–6.

[45] A. Catala, J. Jaen, and J. Modioli, "Strategies for accelerating ant colony optimization algorithms on graphical processing units," in *2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, sept. 2007, pp. 492 –500.

[46] J. Li, X. Hu, Z. Pang, and K. Qian, "A parallel ant colony optimization algorithm based on fine-grained model with GPU-acceleration," *International Journal of Innovative Computing, Information and Control*, vol. 5, no. 11, pp. 3707–3716, 2009.

[47] Y. You, "Parallel ant system for traveling salesman problem on GPUs," in *Eleventh annual conference on genetic and evolutionary computation*, 2009.

[48] J. Cecilia, J. Garcia, M. Ujaldon, A. Nisbet, and M. Amos, "Parallelization strategies for ant colony optimisation on GPUs," in *25th IEEE International Parallel and Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW 2011)*, may 2011, pp. 339 –346.

[49] W. Jiening, D. Jiankang, and Z. Chunfeng, "Implementation of ant colony algorithm based on GPU," in *Sixth International Conference on Computer Graphics, Imaging and Visualization, 2009 (CGIV '09)*, aug. 2009, pp. 50 –53.

[50] H. Bai, D. Ouyang, X. Li, L. He, and H. Yu, "MAX-MIN Ant System on GPU with CUDA," in *Fourth International Conference on Innovative Computing, Information and Control (ICICIC 2009)*, dec. 2009, pp. 801 –804.

[51] J. Fu, L. Lei, and G. Zhou, "A parallel ant colony optimization algorithm with GPU-acceleration based on all-in-roulette selection," in *Third International Workshop on Advanced Computational Intelligence (IWACI 2010)*, aug. 2010, pp. 260 –264.

[52] S. Chen, S. Davis, H. Jiang, and N. A., "CUDA-based genetic algorithm on traveling salesman problem," in *Computers and Information Science*, R. Lee, Ed. Springer Berlin Heidelberg, 2011, pp. 241–252.

[53] J. Li, L. Zhang, and L. Liu, "A parallel immune algorithm based on fine-grained model with GPU-acceleration," in *Fourth International Conference on Innovative Computing, Information and Control (ICICIC 2009)*. IEEE, 2009, pp. 683–686.

[54] E. Taillard, "Benchmark for basic scheduling problems," *Journal of Operational Research*, vol. 64, pp. 278 – 285, 1993.

[55] T. V. Luong, N. Melab, and E. Talbi, "Large neighborhood local search optimization on Graphics Processing Units," in *24th IEEE International Parallel & Distributed Processing Symposium, Workshops and Ph.D. Forum (IPDPSW 2010), Workshop LSPP 2010*. IEEE, 2010.

[56] E. L. Lawler, J. K. Lenstra, A. R. Kan, and D. B. Shmoys, *The traveling salesman problem: a guided tour of combinatorial optimization*. Wiley New York, 1985, vol. 3.

[57] G. Reinelt, "Tsplib—a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[58] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *Computational Intelligence Magazine, IEEE*, vol. 1, no. 4, pp. 28–39, 2006.

[59] G. Laporte and S. Martello, "The selective travelling salesman problem," *Discrete Applied Mathematics*, vol. 26, no. 2, pp. 193–207, 1990.

[60] T. Scavo, "Scatter-to-gather transformation for scalability," Aug 2010. [Online]. Available: https://hub.vscse.org/resources/223

[61] AccelerEyes. Jacket toolbox. [Online]. Available: http://wiki.accelereyes.com/wiki/index.php/JACKET

[62] NVIDIA. (2013) Kepler architecture. [Online]. Available: http://www.nvidia.com/object/nvidia-kepler.html

[63] CAPS, CRAY, NVIDIA, and PGI. (2013) OpenACC Directives for Accelerators. [Online]. Available: http://www.openacc-standard.org/